Lasse Berglund

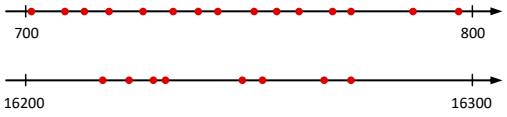
Primtal – det luktar Python

Redan tidigt i sin skolgång bör elever få kännedom om begreppet primtal. Det går sedan att återkomma till begreppet med olika infallsvinklar, exempelvis att med programmering undersöka förekomsten av primtalstvillingar.

Primtalens plats i de nya kursplanerna har tonats ner, och begrepp som kvot och rest har sedan länge befunnit sig i skuggan. På följande sidor tar vi ändå detta stoff som utgångspunkt för lite programmering, ett kursmoment som enligt propåerna ska vara uppe på tapeten.

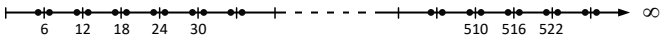
Något slags mönster om än trivialt

Primtalen har i alla tider fascinerat; de har liknats vid talens elementarpartiklar då de inte kan brytas ner i mindre faktorer. Redan *Erathostenes* visste att deras förekomst utmed tallinjen på individnivå är helt oregelbunden. Figur 1 visar två exempel på detta, där de röda prickarna utgör samtliga primtal i det aktuella intervallet.



Figur 1

I figur 2 är 6:ans tabell markerad. Prickarna intill, $6k \pm 1$, är självklart inte primtal allihop. Däremot är det bara bland dessa regelbundet utlagda prickar som vi kan finna några primtal. Det är bara bland dessa tal det är lönt att leta.



Figur 2

Ett heltal som divideras med 6 kan få resterna 0, 1, 2, 3, 4, 5. Så alla heltal a kan skrivas på en av följande former:

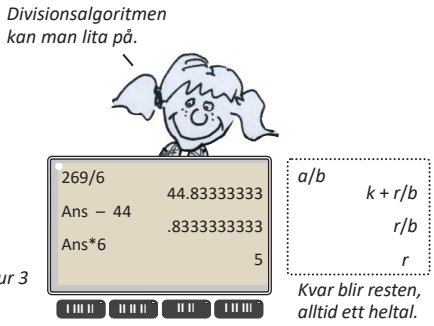
$a = 6k$
$a = 6k + 1$
$a = 6k + 2 = 2(3k + 1)$
$a = 6k + 3 = 3(2k + 1)$
$a = 6k + 4 = 2(3k + 2)$
$a = 6k + 5$

Det framgår att fyra av de sex formerna alltid är sammansatta tal. Alltså gäller prickarnas placering i figur 2. Ett primtal som är större än 3 får alltid resten 1 eller 5 vid division med 6.

En elev vill kanske göra en kontroll av detta. Talet 269 är ett primtal. Med räknaren finner eleven resten enligt följande: *utför divisionen, subtrahera bort heltalsdelen, multiplicera med nämnaren*. Att detta alltid fungerar beror så klart på vår kära gamla *divisionsalgoritm* som säger att om divisionen a/b har kvoten k och resten r, båda heltal, så gäller:

$$a = bk + r \qquad 0 \le r < b$$

där det väsentliga är att r är strängt mindre än b, se figur 3.



Figur 3

Historisk anekdot

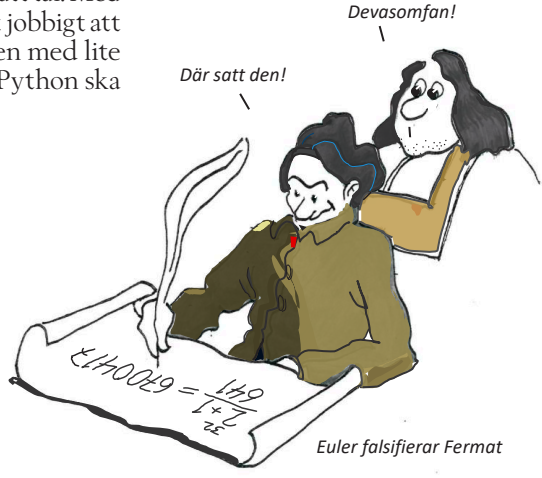
Till en av de stora i matematikens historia räknas *Fermat*, verksam under 1600-talet. Men även han tog miste en gång, och det var då han förmodade att var gång ett positivt heltal n sätts in i formeln

$$F_n = 2^{2^n} + 1$$

så genereras ett primtal. På 1700-talet visade *Euler* att så inte är fallet. Han lyckades med bedriften att visa att

$$F_5 = 2^{2^5} + 1 = 641 \cdot 6\,700\,417$$

det vill säga ett sammansatt tal. Med enbart en penna blir det jobbigt att följa i Eulers fotspår, men med lite basal programmering i Python ska det nog gå.



Grundskolan

Man kan starta med ett tal som 2873. Efter att ha dividerat 2873 med alla tal fram till och med 12 med sina räknare, finner eleverna att det var gång har blivit ett decimaltal. Men så: 2873/13 = 221, ett heltal. Alltså var 2873 = 13 · 221, det vill säga inget primtal. Så här pass enkla betraktelser kan ligga till grund för en primtalskod ämnad för grundskolan.

En kort och enkel kod

Kod 1 Primtal för grundskolan

```
1  a = int(input("Ange ett heltal: "))
2  for  b in range(2,a):
3       if a%b == 0:
4            print b, a/b
5  else:
6            print a, "= primtal"
```

```
Ange ett heltal:  14761
29   509
```

1 En användare har knappat in talet $a = 14\,761$.

2 Med en for-loop ska en kontroll av detta tal nu ske 14 760 gånger.

3 Att en division av två tal a och b går jämnt upp är samma sak som att resten r blir noll. Exempel:
$35/5 = 7 \Leftrightarrow 35 = 5 \cdot 7 \Leftrightarrow 35\%5 == 0$
till skillnad från
$38/5 = 7{,}6 \Leftrightarrow 38 = 5 \cdot 7 + 3 \Leftrightarrow 38\%5 == 3$.
Så om $14\,761\%b == 0$ för något heltal b i for-loopen, innebär det att b är en faktor i 14 761 och därmed är 14 761 inget primtal.

4 Det visar sig att 14 761/29 får resten noll, så det som skrivs ut blir 29 och 509.

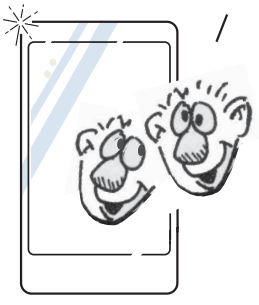
6 Annars skrivs det ut att a är ett primtal eftersom a då har klarat sig genom hela processen.

Kanske någon i klassen får in en jackpot

Två primtal som bara har ett tal emellan sig kallas för *primtalstvillingar*: 857 och 859 är ett exempel. Här är en uppgift som kanske kan roa någon elev:

> Är ditt mobilnummer ett primtal? Använd programmet och kolla. Om så är fallet, gå vidare för en eventuell jackpot: se om ditt mobilnummer även är ena delen av en primtalstvilling.

Twin prime någon?



Gymnasiet

Kanske viktigare än att en kod är kort är att körningarna blir snabba. Kod 1 kontrollerar varenda nämnare ända upp till talet a. Detta är onödigt.

Av Erathostenes har vi lärt oss att det räcker att skutta fram på primtalen för att få veta om a är ett primtal. Multiplarna kan ju inte bidra med något nytt. Här ska vi inte programmera alltför avancerat, men kan vi slippa att kontrollera alla jämna tal så är mycket vunnet.

Dessutom: har man inte funnit någon faktor innan $\sqrt{a}$ behöver man inte leta längre. En förklaring i klassen till detta kan vara: två faktorer i ett tal a kan inte båda vara större än $\sqrt{a}$. Exempel: $64 = 8 \cdot 8$ som självklart också kan skrivas som $4 \cdot 16$ med den ena faktorn större än 8. Men båda faktorerna kan inte vara större än $8 = \sqrt{64}$.

Färre varv i loopen

6 Så länge ett tal b är mindre än $\sqrt{a}$ ska följande ske:

7 Om b går jämnt upp i a,

8 ska de två aktuella faktorerna skrivas ut.

9 Talet b tilldelas därpå värdet a så att det blir större än $\sqrt{a}$. Villkoret för while-loopen blir då överspelat.

11 Annars ökas b på med 2 så att vi hamnar på nästföljande udda tal som nu ska in i if-kontrollen ovan.

12 Om b är mindre än a har vi aldrig kommit ur while-loopen, det vill säga någon rest noll har aldrig visat sig.

13 Då är a ett primtal och detta skrivs ut.

Kod 2 Primtal för gymnasiet

```
1  a = int(input("Ange ett heltal > 3: "))
2  b = 3
3  if a%2 == 0:
4      print 2, a/2
5  else:
6      while b <= a**0.5:
7          if a%b == 0:
8              print b, a/b
9              b = a
10         else:
11             b = b + 2
12     if b < a:
13         print a, "= primtal"
```

```
Ange ett heltal > 3:  2**32+1
641     6700417
```

Körningen visar det som redan Euler visste: F_5 är inget primtal.

Några jämförelser

Talet 15 761 453 är ett primtal. Det tar cirka tre sekunder för kod 1 att konstatera detta och för kod 2 tar det cirka 0,1 sekunder. För kod 1 blir det fler än 15 miljoner varv i for-loopen. För kod 2 får vi $\sqrt{15\,761\,453} \approx 3\,970$. Fram till detta tal ska vi bara kontrollera vartannat tal, vilket ger 1 985 varv i for-loopen. Dessa jämförelser kan kanske inspirera en del elever till att ytterligare trimma sina koder. Diskussioner kan uppstå.

Två övningar man kan klara utan program

> Talet 79 523 består av två faktorer och dessa är primtalstvillingar. Vilka?

En gymnasieelev kan klara detta då det blir en andragradsekvation. En högstadieelev kan kanske inse att de två faktorerna måste vara ungefär lika stora. Så med räknaren finner eleven att talen bör ligga i närheten av $\sqrt{79\,523} \approx 281{,}9$ och börjar därför lämpligen leta i de trakterna.

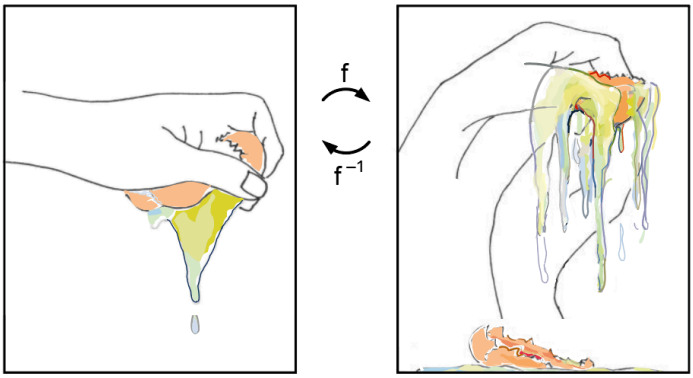
> Utan program och även utan räknare: talet 3 746 består av två primfaktorer. Ange den minsta faktorn.

Som att knäcka ägg fast tvärtom

Modern kryptering bygger på insikten om att det är svårt att hitta primfaktorerna i ett stort tal. Det motsatta gäller inte. Att multiplicera ihop två stora tal är lätt. Det är bara att knappa in talen på datorn. Denna omständighet benämns ibland *envägsfunktion*. Lätt som en plätt åt ena hållet, oerhört svårt åt det andra. Om Pelle får uppgiften att knäcka ett ägg, och Lisa sen den motsatta, att ur stekpannan plocka upp allt och få ihop ägget igen, så har vi en bild av en envägsfunktion.

Låt eleverna knappa in en produkt på sina räknare; vilken som helst. Det tar dem inte mer än några sekunder. Skriv därefter upp ett sexsiffrigt tal på tavlan som består av två primfaktorer och låt de med sina räknare sitta och leta en stund för att finna dessa faktorer. Detta är att samla ihop det krossade ägget. När de sen till slut tröttnar, kan de ju använda sina program så klart och direkt finna det de söker. Vad har då detta med kryptering att göra då programmen gör det på bråkdelen av en sekund? Om det givna talet inte består av sex siffror som ovan utan av säg trehundra siffror, tar det en modern dator ett par tre sekel tuggande miljontals varv per sekund att finna de två primfaktorerna.

Låt eleverna med sina program leta fram egna primtal och av dem bilda produkter som de sedan kan ge till varandra, ungefär som hemliga meddelanden. Kanske ett intresse för kryptering kan slå rot hos någon. Kanske någon blir lockad till att fortsätta studera matematik.



Att knäcka ett ägg är väl ingen konst.

Men utför inversen den som kan.

HISTORIK

Eratosthenes, 200-talet f Kr, med primtalssållet.
Pierre de Fermat, 1607–1665, med grunderna till talteorin.
Leonhard Euler, 1707–1783, med stora bidrag till matematikens alla områden.